Android 应用程序中代码异味共存现象的实证研究 *

边奕心，王露颖†，赵　松，朱　晓

(哈尔滨师范大学 计算机科学与信息工程学院, 哈尔滨 150025)

摘　要： 相对于单一类型的代码异味，代码异味共存现象更具危害性。已有实证研究大多聚焦于分析桌面应用程序中代码异味的共存现象，缺少对 Android 应用程序中代码异味共存现象的研究。为了研究 Android 应用程序中代码异味的共存现象，并与桌面应用程序中代码异味共存现象进行比较，分别对 285 个 Android 应用程序和 30 个桌面应用程序进行检测，对检测出来的 10 种异味进行分析。分析方法如下：首先，根据检测结果计算受到多种异味影响的类的百分比。然后，使用公式计算代码异味共存的频率，最后，使用 Spearman 相关系数分析代码异味共存与应用程序规模的关系。结论如下：a)在 Android 应用程序中受到一种以上代码异味共同干扰的类占有异味的类的总数的 31.04%；b)在两个平台的应用程序中，两对代码异味 Brain Class—Brain Method 和 God Class—Brain Method 共存的频率较高；c)一种异味、两种异味共存、三种异味共存与 Android 应用程序的规模具有较强的相关性。

关键词： 代码异味共存；Android 应用程序；桌面应用程序；实证研究

中图分类号： TP31　　doi: 10.19734/j.issn.1001-3695.2022.02.0060

Empirical study on code smell co-occurrences in Android applications

Bian Yixin, Wang Luying†, Zhao Song, Zhu Xiao

(*College of Computer Science & Information Engineering*, *Harbin Normal University*, *Harbin* 150025, *China*)

Abstract: Code smell co-occurrences are more harmful to the systems than the individual smell type. Although multiple research works have focused on the code smell co-occurrences in traditional desktop applications, the researchers pay less attention to the code smell co-occurrences in Android applications. In order to study the phenomenon of Object-Oriented code smells co-occurrences in Android applications and then compares it with the phenomenon of code smell co-occurrences in desktop applications. It analyzed 10 Object-Oriented code smell types detected in 285 Android apps and 30 desktop apps. First, the percentage of classes affected by various kinds of smells is calculated according to the detection results. Second, the frequency of code smell co-occurrences is analyzed with a formula. Finally, the Spearman correlation coefficient is used to analyze the relationship between code smell co-occurrences and the size of the application. The results show that: a) in Android applications, 31.04% of smelly classes are affected by more than one type of code smells. b) In the applications of the two platforms, two pairs of code smells, Brain Class -Brain Method and God Class-Brain Method, frequently co-occur. c) One type of smells, the co-occurrences of two types of smells, and the co-occurrences of three types of smells are strongly correlated with the size of Android applications.

Key words: code smell co-occurrences; Android applications; desktop applications; empirical study

0　引言

代码异味(code smells)又被称为代码坏味道，是软件系统中存在设计缺陷的代码段[1]。代码异味共存(code smell co-occurrences)是指一个类中存在一种以上的代码异味。研究表明，相对于单一类型的代码异味，代码异味共存对系统更具威胁性[2]。Palomba 等人[3]对传统桌面应用程序中的代码异味共存现象进行了实证研究，结果显示代码异味共存现象普遍存在于桌面应用程序中，而且在受异味干扰的类中，最多包含三种不同类型的代码异味。

近年来，随着移动通信技术的迅猛发展，移动应用程序已经成为软件行业的发展主体。从 2016 年开始，全球移动应用程序下载量持续增长，2019 年下载量超过 2000 亿次。2021 年全球移动应用的下载量已经达到 2300 亿次，较 2020 年增加了 120 亿次。其中，Android 应用程序的市场占有率占手机应用程序的 80%以上，而 iOS 平台上的应用程序只占不到 20%。研究表明代码异味可以影响任何软件系统[4,5]，已有实证研究大多聚焦于分析桌面应用程序中代码异味的共存现象，缺少针对 Android 应用程序中代码异味共存现象的研究。因此，目前尚不清楚在 Android 应用程序中是否存在代码异味的共存现象？如果存在，对系统的影响程度如何？另外，由于传统桌面应用程序与 Android 应用程序在程序结构、API 调用、内存、CPU、网络、电池等方面的诸多差异，Android 应用程序中代码异味的共存现象与传统桌面应用程序中代码异味的共存现象之间会有哪些不同，也是未知？

针对以上问题，本文采用实证研究方法分析 Android 应用程序中代码异味的共存现象，并与传统桌面应用程序中代码异味共存现象进行对比。通过实证研究，结论如下：

a)Android 应用程序中存在代码异味共存现象，其中，受代码异味影响的类中有 31.04% 的类受到一种以上代码异味

收稿日期： 2022-02-06；**修回日期：** 2022-04-17　**基金项目：** 国家自然科学基金项目(61902094)；哈尔滨师范大学博士科研启动基金项目(XKB201801)；黑龙江省自然科学基金项目(QC2018082)；黑龙江省普通本科高等学校青年创新人才培养计划资助项目(UNPYSCT-2018183)；哈尔滨师范大学计算机科学与信息工程学院科研项目(JKYKYY202004，JKYKYZ202104)；哈尔滨市科技局科技创新人才研究专项资金项目(RC2017QN010002)

作者简介： 边奕心(1979-)，女，吉林吉林人，讲师，硕导，博士，主要研究方向为软件重构、程序分析；王露颖(1998-)，女(通信作者)，黑龙江牡丹江人，硕士，主要研究方向为软件重构、程序分析(bianyu79@163.com)；赵松(1976-)，男，黑龙江哈尔滨人，讲师，硕士，主要研究方向为计算机体系结构、程序分析，代码重构；朱晓(1984)，男，山东人，讲师，硕导，博士，主要研究方向为生物信息处理.

的共同干扰。在这些含有共存异味的类中，0.05%的类受到 6 种代码异味的共同干扰，因此，在 Android 应用程序中，虽然受到多种代码异味共同干扰的类的比例并不大，但这种现象对系统的危害程度却很严重。相对于 Android 应用程序，桌面应用程序中的代码异味共存现象则更为普遍，其对系统的危害程度也高于 Android 应用程序。

b) 在 Android 应用程序中，两对代码异味 Brain Class—Brain Method 和 God Class—Brain Method 共同发生的频率高于其他异味。在桌面应用程序中，三对代码异味 Brain Class—Brain Method、God Class—Brain Method 和 Refused Parent Bequest—Brain Method 共同发生的频率高于其他异味。由此可见，两对代码异味 Brain Class—Brain Method 和 God Class—Brain Method 在两个平台的应用程序中都易于共存。此外，两个平台的应用程序中，共存的代码异味之间并不是一一对应的关系。

c)代码异味共存现象与 Android 应用程序的规模存在相关性。即应用程序的规模越大，其中含有一种异味的类数量、同时含有两种异味的类数量和同时含有三种异味的类数量越多，而同时含有四种、五种和六种异味的类数量与程序规模不相关。

以上研究结论有助于 Android 应用程序的研究者和工具开发者加深对代码异味的认识，从而更有效的处理代码异味共存对 Android 应用程序的负面影响，降低维护成本，提高 Android 应用程序的质量。对于研究人员，异味之间的共存关系有助于异味检测与重构的研究，研究人员在研究异味检测时不仅要考虑单个异味的检测方法，还要考虑异味之间的依赖关系，研究异味共存的检测方法。此外，研究人员在对 Android 应用程序的异味进行重构时，不仅要考虑单个异味的重构方式，还要考虑异味之间的依赖关系。利用这种依赖关系研究异味的调度顺序，从而节省重构的时间，提高重构的准确性和效率，最终目的是提高 Android 应用程序的质量。对于工具的开发者，可以利用异味之间的依赖关系进行异味的重构，即最先重构那些共存频率较高的异味，从而节省开发的时间成本。

1　相关研究

1.1　Android 应用程序中的代码异味

Fowler 等人[1]最先提出了 22 种代码异味并给出相应的重构方法。随着移动设备的迅速发展，移动应用程序已经成为软件行业的发展主体。许多学者的关注点已经从传统的桌面应用程序转移到 Android 应用程序。Mannan 等人[6]通过对比 Android 应用程序和桌面应用程序中的代码异味，通过实证方法研究不同平台下代码异味的差异。Rahkema 等人[7]通过对比 iOS 应用程序和 Android 应用程序发现，除了一种代码异味 Distorted hierarchy 之外，在 Android 应用程序中存在的代码异味也发生在 iOS 应用程序中。

研究表明，不同于传统的桌面应用程序，Android 应用程序中还存在一些 Android 特有的代码异味。Reimann 等人[8]提出了 30 种 Android 特有的代码异味，这些异味主要对 Android 应用程序的非功能属性产生干扰。继 Reimann 之后，越来越多的学者从不同角度对 Android 特有代码异味展开研究。Habchi 等人[9]通过分析 8 种不同的 Android 特有代码异味，研究其产生的原因，进化规律和移除方法。Rasool 等人[10]开发了一个 Android 特有代码异味的检测工具 DAAP，这个工具可以检测出 25 种 Android 特有的代码异味。本文研究 Android 应用程序中面向对象代码异味的共存现象，不包括 Android 特有代码异味。

以上文献都是针对单个类型的代码异味展开研究，缺少对 Android 应用程序中代码异味共存现象的研究。

1.2　代码异味共存

代码异味共存是指在同一个类或方法中，包含两种或两种以上的代码异味[11]。Pietrzak 等人[12]最早发现了代码异味共存现象，他们定义了六种代码异味共存关系，并对共存的代码异味之间的依赖关系进行了深入研究。Yamashita[13]和 Sjøberg[14]等人认为代码异味的共存降低了系统的可维护性，需要通过重构解决问题。Palomba 等人[2,3]对 30 个桌面应用程序的 395 个软件版本进行了实证研究，结果表明代码异味共存现象普遍存在于桌面应用程序中，并且，与受一种异味影响的类相比，受多种代码异味共同影响的类更易发生改变和产生错误。Martins 等人[11]通过分析 3 个 Java 闭源软件的 11 个版本，研究代码异味共存现象对系统内部质量属性的影响。结果显示，通过重构消除代码异味的共存可以降低系统的复杂性。以上文献都是研究传统的桌面应用程序中代码异味的共存现象。

Hamdi 等人[15] 对 Android 应用程序中的代码异味共存现象进行了实证研究，他们分析了 15 种类型的 Android 特有代码异味和 10 种面向对象代码异味。结果表明，代码异味共存现象在 Android 应用程序中比较普遍，并列出了 14 对共存频率较高的代码异味对。尽管 Hamdi 等人综合考虑了两种类型的代码异味共存，但是，他们的研究还存在以下不足：首先，他们并未对检测工具的输出结果进行人工复检，因此，研究结果取决于工具的输出结果。而通常情况下，代码异味检测工具都会存在一定程度的误检和漏检现象[15]。其次，他们并未探讨代码异味共存现象与 Android 应用程序的规模之间的相关性。是否代码量越大，共存的异味就越多？此外，由于 Android 应用程序与传统桌面应用程序的诸多差异，这种共存现象在不同平台的应用程序中会发生怎样的变化？Hamdi 等人并未涉及该问题。而 Mannan 等人[6]的研究表明，对比研究两种平台的代码异味之间的差异性具有实际意义。最后，共存的异味之间是简单的一一对应关系还是存在其他关系，Hamdi 等人并未说明？这种关系直接影响到后续异味的重构研究，因为代码异味检测的目的是为了最终移除异味，提高程序的质量。

针对以上问题，本文采用实证研究方法分析 Android 应用程序中代码异味的共存现象，并与传统桌面应用程序中代码异味共存现象进行对比。相对于单一的代码异味，异味之间的共存反映了异味之间的依赖关系，分析这种依赖关系，可以加深人们对代码异味的认识，更好的理解异味产生的原因，从而在开发之初，有意识的避免引入代码异味。在软件维护阶段，可以利用异味间的依赖关系，研究检测异味共存的方法及重构方法，从而最大限度降低代码异味对系统的危害。

2　实证研究

2.1　研究问题

本文通过分析 Android 应用程序的源代码，研究 Android 应用程序中代码异味的共存现象，围绕这个目标，提出以下 3 个研究问题(RQs)：

RQ1：在 Android 应用程序中是否存在代码异味共存现象？若存在，其共存程度如何？与桌面应用程序中的代码异味共存程度相比，有何不同？

RQ2：在 Android 应用程序中，哪几种代码异味更倾向于共存？这种情况与桌面应用程序中的代码异味共存相比，有哪些不同？

RQ3：代码异味共存现象与 Android 应用程序的规模之间是否存在相关性？

2.2　实验对象

2.2.1 应用程序的选取

本文选取 285 个 Android 应用程序和 30 个桌面应用程序

序作为实验对象。其中，285 个 Android 应用程序的选择标准如下：

a) 首先，在 AndroZooOpen[16]中选出 539 个开源 Android 应用程序。AndroZooOpen 是一个开源的数据集，目前包含 76466 个 Android 应用程序。AndroZooOpen 中包含由不同语言开发的 Android 应用程序，本文选择 Java 语言开发的程序。原因如下：Java 是目前移动应用软件开发的主流语言之一，并且针对 Java 的代码异味检测工具较多且易于获取。

b) 然后，使用开源工具 RepoReapers[17]去除低质量的 Android 项目。RepoReapers 可以从 8 个方面对数据集中的程序质量进行评分(分别是体系结构、社区、持续集成、文件、提交历史、许可证、问题和单元测试)。如果待选程序在 7 个方面的评分均大于零，则选取该程序。经过 RepoReapers 筛选之后，选出 321 个 Android 应用程序作为候选程序。

c) 最后，为了保证所选 Android 应用程序种类的多样性，本文根据 Google Play Store 对应用程序的分类[18,19]将待选 Android 项目分为 6 个类别，分别是：开发、影音、社交通信、家庭和教育、安全和通用以及其他，每种类别所占比例如图 1 所示。

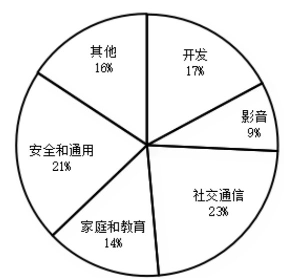


图 1　285 个 Android 应用程序所属类别及比例

Fig. 1　Categories and proportions of 285 Android applications

最终，经过以上筛选过程，最后收集了 285 个 Android 应用程序作为实验对象，共计 34617 个类、315634 个方法、2701826 行代码。此外，还需选取一定数量的桌面应用程序作为对比研究的另一实验对象。本文选取 Palomba 等人[3]在研究传统桌面应用程序中代码异味共存现象时，所使用的 30 个开源数据集，共计 43257 个类、436299 个方法以及 4948368 个方法。

2.2.2 代码异味检测工具

本文使用代码异味检测工具 iPlasma 对所选应用程序进行异味检测。目前，主流的代码异味检测工具主要有以下七个，iPlasma、inFusion、Checkstyle、JDeodorant、PMD、DECOR 和 Stench Blossom[20]。Fontana[20]等人的研究结果表明，在这七个检测工具中 iPlasma 和 inFusion 是可检测代码异味种类最多的两个工具。Mannan 等人[6]的研究表明，用于检测桌面应用的检测工具，同样可检测 Android 应用程序中的代码异味。尽管 inFusion 声称可以检测出 Fowler 提出的 22 种代码异味，但是，Rahkema 等人[7]的研究表明，inFusion 代码异味检测工具目前已经无法获取。此外，Reis 等人[21]的研究结果显示，5.8%的研究使用 inFusion 进行检测，15.4%的研究使用 iPlasma 对代码异味进行检测。因此，本文选用 iPlasma 检测异味。

2.2.3 代码异味的选取

本文选取的 10 种代码异味如表 1 所示。这 10 种代码异味经常被用于异味的相关研究中[6]，并且本文使用的检测工具 iPlasma 可以检测这 10 种代码味道。

2.3　实验流程

实验过程如图 2 所示。

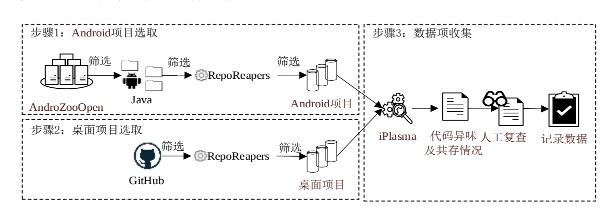


图 2　代码异味及共存情况收集过程

Fig. 2　Code smell and co-occurrences collection process

在确定好实验对象后，执行以下步骤：

a) 代码异味检测：使用代码异味检测工具 iPlasma 分别对 2.2.1 节选出的 285 个 Android 应用程序和 30 个桌面应用程序进行检测。

b) 手动检查检测结果：尽管 iPlasma 的检测结果准确度很高，但是，由于其检测规则本身的局限性，会产生一定数量的误检。因此，本文采用人工复检方法，对工具输出的代码异味进行筛查。招募了 4 名志愿者，其中，1 名为计算机专业的硕士研究生，3 名为计算机专业的本科生。将 4 名同学分为两组，其中，2 名同学独立对 285 个 Android 应用程序中代码异味的检测结果进行检查，另外 2 名同学独立对 30 个桌面应用程序中的代码异味的检测结果进行检查。人工复查具体步骤如下：

(a) 对 4 名同学进行一定的培训，使其对代码异味有较深入的认识，理解并可以根据研究所涉及的 10 种代码异味的相关描述，准确地找到应用程序中所存在的异味；

(b) 每名同学逐一检查所有原文件，参照代码异味定义，逐一检查源代码文件中的代码异味，并记录结果；

(c) 当两组同学均完成人工分析后，使用 Cohen's kappa[22]系数分析每组中的两位同学对异味检测的一致性。本研究中 Cohen's kappa 系数 0.63；。

(d) 与工具输出结果及报告的异味所在文件路径进行比对，并对存在意见不一致的结果进行复检，达成一致意见后，记录数据。

c) 记录代码异味检测结果：根据人工复检结果，记录代码异味相关信息，例如其所在类\方法名等。再将存在共存的类\方法单独记录，用于后续研究。

表 1　代码异味的描述

Tab. 1　Description of code smells

#	代码异味	描述
1	God Class(GC)	一个大而复杂的类，它通常集中了许多功能
2	Data Class(DC)	一个只有状态没有行为的类
3	Brain Class(BC)	一个集中过多功能的类，通常受到 Brain Method 的影响
4	Feature Envy(FE)	当一个方法对其他类的特性比它自己类的更感兴时，就会出现这种异味
5	Brain Method(BM)	倾向于集中整个类的功能的方法
6	Intensive Coupling(IC)	从其他类调用多个方法的方法
7	Extensive Coupling(EC)	调用多个类的一个或多个方法的方法
8	Shotgun Surgery(SS)	系统一个地方的改变，涉及到了许多其他地方的相关变化
9	Tradition Breaker(TB)	当派生类打破其基类为系统的其他部分提供的服务的继承原则，并提供了一组与其基类提供的服务无关的服务时，就会出现这种异味
10	Refused Parent Bequest(RPB)	派生类很少或没有使用其基类定义的特定于继承的成员时，就会出现这种异味

2.4 分析方法

实验一：为了回答 RQ1，首先根据检测结果计算出每个类中包含的代码异味数量。然后，计算受一种代码异味和多种代码异味影响的类的百分比。

实验二：RQ2 主要研究 Android 应用程序中代码异味共存的频率。本文使用 Palomba 等人提出的公式[3]计算异味的共存频率，如式(1)所示。

$$co-occurrences_{cs_{i,j}} = \frac{|cs_i \wedge cs_j|}{|cs_i|}, \text{且} i \neq j \quad (1)$$

其中，CS_i 和 CS_j 分别表示不同种类的代码异味，分子 $|CS_i \wedge CS_j|$ 表示代码异味 CS_i 和 CS_j 共同发生的总次数，分母 $|CS_i|$ 表示代码气味 CS_i 出现的总次数。如果将公式中的 i 和 j 交换后，所得的结果是不同的。

实验三：为了回答 RQ3，本文使用 Spearman 相关性系数分析代码异味共存现象与 Android 应用程序规模之间的关系。这里，使用类的数量、方法的数量和代码行数表示 Android 应用程序的规模。本文使用 Cohen 提供的一套解释相关系数的指南[20]，用以判断代码异味共存现象与 Android 应用程序规模之间相关性的强弱程度。

3 研究结果与分析

RQ1：表 2 列出了两个平台中代码异味共存的情况。根据表 2 的结果可以看出：在 Android 应用程序中同样存在代码异味共存现象。其中，受到一种以上代码异味共同干扰的类占含有异味的类的总数的 31.04%，在这些含有共存异味的类中，21.73%的类受到两种代码异味的干扰，占比最大。此外，受到三种、四种、五种以及六种代码异味共同干扰的类分别占 6.64%、2.21%、0.37%和 0.09%。从表 2 中还可以看出 Android 应用程序中代码异味共存程度较桌面应用程序要高出 1.34%(=31.04%-29.70%)。Palomba 等人[14]通过对 30 个桌面应用程序的 395 个软件版本进行实证研究，通过分析其中的 13 种代码异味发现在桌面应用程序中最多只受到三种代码异味共存的影响。而本则在 30 个桌面应用程序中发现了四种、五种、六种代码异味共存的情况。

RQ2：表 3 列出了 10 种代码异味在 Android 应用程序中共同发生的频率。由表 3 可知，在 Android 应用程序中，两对代码异味共存的频率高于其他异味，这两对异味是：Brain Class—Brain Method 和 God Class—Brain Method。其中，受到 Brain Class 干扰的类中有 61%的类同时受到异味 Brain Method 的干扰，受到 God Class 干扰的类中有 17%的类同时受到异味 Brain Method 的干扰。 此外，共存的代码异味之间并不是一一对应的关系，例如，在 Android 应用程序 toposuite 的一个类中，存在 8 个 Feature Envy 实例和 8 个 Intensive Coupling 实例。

表 4 列出了 10 种代码异味在桌面应用程序中共同发生的频率。如表 4 所示，在桌面应用程序中，Brain Method 更倾向于和其他代码异味共同发生。其中，三对代码异味发生共存的频率高于其他异味，这三对异味是：Brain Class—Brain Method、God Class—Brain Method 和 Refused Parent Bequest—Brain Method。此外，与 Android 应用程序相似，这些共存的代码异味之间并不是一一对应的关系。

表 2 受到一种至六种代码异味干扰的类的占比

Tab. 2 Proportion of classes that are disturbed by oneto six code smells

异味干扰的类	Android		桌面	
	总数	占比	总数	占比
受一种代码异味影响的类	5376	68.96%	5718	70.30%
受两种代码异味影响的类	1694	21.73%	1263	15.53%
受三种代码异味影响的类	518	6.64%	691	8.50%
受四种代码异味影响的类	172	2.21%	307	3.77%
受五种代码异味影响的类	29	0.37%	124	1.52%
受六种代码异味影响的类	7	0.09%	31	0.38%

表 3 在 Android 应用程序中代码异味共同发生的频率

Tab. 3 The frequency of code smell co-occurring in Android applications

i/j	GC (1178)	DC (2582)	BC (295)	FE (3854)	BM (2474)	IC (1332)	EC (247)	SS (1627)	TB (87)	RPB (123)
GC (1178)		0%	0%	**16%** (187)	**17%** (205)	3% (41)	1% (14)	4% (49)	1% (7)	1% (13)
DC (2582)	0%		0%	3% (66)	2% (52)	1% (29)	0%	3% (68)	0%	0% (6)
BC (295)	0%	0%		0%	**61%** (181)	0%	0%	0%	0%	0%
FE (3854)	5% (184)	2% (66)	0%		3% (132)	3% (107)	1% (18)	0% (9)	0%	0% (5)
BM (2474)	8% (207)	2% (54)	7% (183)	5% (132)		5% (121)	1% (22)	1% (13)	0%	0%
IC (1332)	3% (41)	2% (29)	0%	8% (104)	10% (128)		0% (5)	0% (6)	1% (13)	0%
EC (247)	4% (9)	0%	0%	6% (15)	9% (22)	2% (4)		2% (4)	0%	0%
SS (1627)	3% (49)	4% (68)	0%	1% (10)	1% (14)	0% (3)	0% (4)		0% (4)	0%
TB (87)	6% (5)	0%	0%	0%	0%	**13%** (11)	0%	5% (4)		**30%** (26)
RPB (123)	10% (12)	3% (4)	0%	3% (4)	0%	0%	0%	0%	**15%** (19)	

注：表格中的字体加粗数据，是共同发生频率>10%的代码异味

表 4　桌面应用程序中代码异味共同发生的频率

Tab. 4　The frequency of code smell co-occurring

i/j	GC (1095)	DC (2064)	BC (648)	FE (1025)	BM (1967)	IC (1839)	EC (874)	SS (1817)	TB (112)	RPB (426)
GC (1095)		0%	0%	6% (62)	**15%** (164)	3% (37)	2% (23)	6% (68)	0%	1% (16)
DC (2064)	0%		0%	1% (28)	5% (96)	2% (46)	2% (41)	3% (67)	0%	0% (8)
BC (648)	0%	0%		0%	**51%** (329)	0%	0%	0%	0%	0%
FE (1025)	6% (64)	3% (31)	0%		10% (96)	**12%** (123)	2% (19)	1% (12)	0% (4)	1% (11)
BM (1967)	8% (164)	5% (92)	5% (92)	1% (27)		3% (59)	1% (18)	0% (7)	0%	1% (21)
IC (1839)	1% (22)	1% (26)	0%	4% (65)	7% (121)		0% (5)	0%	0%	1% (13)
EC (874)	1% (11)	3% (24)	0%	2% (14)	4% (33)	1% (7)		0%	0%	0% (3)
SS (1817)	1% (19)	1% (16)	0%	0% (5)	1% (11)	0%	0%		0% (4)	0%
TB (112)	0%	0%	0%	3% (3)	0%	0%	0%	6% (7)		**10%** (11)
RPB (426)	4% (15)	2% (8)	0%	3% (11)	**17%** (71)	4% (19)	1% (4)	0%	5% (22)	

注：表格中的字体加粗数据，是共同发生频率>10%的代码异味

综合表 3 和 4 可以看出，在两个平台的应用程序中，Brain Class—Brain Method 和 God Class—Brain Method 两对代码异味共存的频率都较高。

RQ3：如表 5 所示，由于在 Android 应用程序中五种、六种代码异味共存的程度均不足 1%。因此，此处不考虑五种和六种代码异味共存现象与 Android 应用程序规模之间的关系。如表 5 所示，一种代码异味、两种代码异味共存、三种代码异味共存现象与 Android 项目规模具有较强的相关性(即项目规模越大，项目中含有一种代码异味的类、同时含有两种代码异味的类和同时含有三种代码异味的类的数量越多)，四种代码异味共存现象与 Android 应用程序的规模具有中等程度的相关性。

表 5　代码异味共存现象与 Android 项目规模的相关性

Tab. 5　The correlation between the code smell co-occurrences and the scale of the Android project

代码异味共存	最多	最少	类		方法		代码行		相关性
			r	p-value	r	p-value	r	p-value	
一种代码异味	127	0	**0.87**	**<0.01**	**0.89**	**<0.01**	**0.88**	**<0.01**	**强**
两种代码异味共存	28	0	**0.79**	**<0.01**	**0.82**	**<0.01**	**0.86**	**<0.01**	**强**
三种代码异味共存	15	0	**0.52**	**<0.01**	**0.51**	**<0.01**	**0.52**	**<0.01**	**强**
四种代码异味共存	7	0	0.38	<0.01	0.49	<0.01	0.50	<0.01	中

4　结束语

本文对 Android 应用程序中代码异味的共存现象进行了深入分析。本文与代码异味共存领域相关研究的最大区别在于，首次对 Android 应用程序中的代码异味共存现象进行深入的实证研究。根据本文实证研究的结果发现，在 Android 应用程序中确实存在代码异味共存现象，其中受到一种以上代码异味共同干扰的类占含有异味的类的总数的 31.04%。通过对比 Android 和桌面两个平台的应用程序，分析其中的代码异味共存现象发现，Brain Class—Brain Method 和 God Class—Brain Method 这两对代码异味共存的频率相对较高。此外，研究还发现异味共存与 Android 应用程序的规模存在相关性。

总体而言，实证研究结果有助于 Android 应用程序的开发人员和维护人员加深对其中代码异味的认识，从而更有效的处理代码异味共存现象对 Android 应用程序的负面影响，降低开发和维护成本，提高 Android 应用程序的质量。但是，本研究还存在以下待改进的空间：还可以选取更多具有代表性的开源 Android 应用程序甚至商用软件；还可以考虑分析更多种代码之间的共存现象等。

在后续的研究中，将扩大实证研究的规模，分析更多的 Android 应用程序，研究代码异味共存现象。以及探索由其他语言(如 Kotlin)开发的项目中是否存在代码异味共存现象及其对程序的影响。

参考文献：

[1] Martin Fowler. 重构: 改善既有代码的设计 [M]. 熊节, 林从羽, 译. 2 版. 北京: 人民邮电出版社, 2019. (Martin Fowler. Refactoring: improving the design of existing code [M]. translated by Xiong jie, Lin congyu. 2nd ed. Beijing: Posts&Telecom Press, 2019.)

[2] Palomba F, Bavota G, Penta D M, *et al.* On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation [J]. Empirical Software Engineering, 2018, 23 (3): 1188-1221.

[3] Palomba F, Bavota G, Penta D M, *et al.* A large-scale empirical study on the lifecycle of code smell co-occurrences [J]. Information and Software Technology, 2018, 99: 1-10.

[4] Macia I, Garcia J, Popescu D, *et al.* Are automatically-detected code anomalies relevant to architectural modularity?An exploratory analysis of evolving systems [C]// Proc of the 11th annual international conference on Aspect-oriented Software Development. 2012: 167-178.

[5] Oizumi W, Garcia A, Sousa L S, *et al.* Code anomalies flock together: Exploring code anomaly agglomerations for locating design problems [C]// Proc of IEEE/ACM 38th International Conference on Software Engineering (ICSE) . IEEE, 2016: 440-451.

[6] Mannan U A, Ahmed I, Almurshed R A M, *et al.* Understanding code smells in android applications [C]// Proc of IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft) . IEEE, 2016: 225-236.

[7] Rahkema K, Pfahl D. Comparison of Code Smells in iOS and Android Applications [C]// Proc of QuASoQ@ APSEC. 2020: 79-86.

[8] Jan Reimann, Martin Brylski, Uwe Assmann. A tool-supported quality smell catalogue for android developers [C]// Proc of the conference Modellierung in the Workshop Modellbasierte und modellgetriebene Softwaremodernisierung–MMSM. 2014, 2014.

[9] Habchi S, Moha N, Rouvoy R. Android code smells: From introduction to refactoring [J]. Journal of Systems and Software, 2021, 177: 110964.

[10] Rasool G, Ali A. Recovering Android Bad Smells from Android Applications [J]. Arabian Journal for Science and Engineering, 2020, 45 (4): 3289-3315.

[11] Martins J S, Bezerra C I M, Uchôa A, *et al.* Are code smell co-occurrences harmful to internal quality attributes?a mixed-method study [C]// Proc of the 34th Brazilian Symposium on Software Engineering. 2020: 52-61.

[12] Pietrzak B, Walter B. Leveraging code smell detection with inter-smell relations [C]// Proc of International Conference on Extreme Programming and Agile Processes in Software Engineering. Springer, Berlin, Heidelberg, 2006: 75-84.

[13] Yamashita A, Moonen L. Exploring the impact of inter-smell relations on software maintainability: An empirical study [C]// Proc of the 35th International Conference on Software Engineering (ICSE) . IEEE, 2013: 682-691.

[14] Sjøberg D I K, Yamashita A, Anda B C D, *et al.* Quantifying the effect of code smells on maintenance effort [J]. IEEE Trans on Software Engineering, 2012, 39 (8): 1144-1156.

[15] Hamid O, Ouni A, AlOmar E A, *et al.* An empirical study on code smells co-occurrences in android applications [C]// Proc of the 36th IEEE/ACM International Conference on Automated Software Engineering Workshops. 2021: 26-33.

[16] Liu Pei, Li Li, Zhao Yanjie, *et al.* Androzooopen: Collecting large-scale open source android apps for the research community [C]// Proc of the 17th International Conference on Mining Software Repositories. 2020: 548-552.

[17] Munaiah N, Kroh S, Cabrey C, *et al.* Curating github for engineered software projects [J]. Empirical Software Engineering, 2017, 22 (6): 3219-3253.

[18] Ahmed I, Mannan U A, Gopinath R, *et al.* An empirical study of design degradation: How software projects get worse over time [C]// Proc of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) . IEEE, 2015: 1-10.

[19] Souza L B L, Maia M A. Do software categories impact coupling metrics? [C]// Proc of the 10th Working Conference on Mining Software Repositories (MSR) . IEEE, 2013: 217-220.

[20] Fontana F A, Braione P, Zanoni M. Automatic detection of bad smells in code: An experimental assessment [J]. Journal of Object Technol. , 2012, 11 (2): 5: 1-38.

[21] Reis J P, Abreu F B, Carneiro G F, *et al.* Code Smells Detection and Visualization: A Systematic Literature Review [J]. Archives of Computational Methods in Engineering, 2021: 1-48.

[22] Cohen, J. A Coefficient of Agreement for Nominal Scales [J]. Educational & Psychological Measurement, 1960, 20 (1): 37-46.